AIPS Memo 84

A Proposed Package to Support the Use of the X Window System in AIPS Tasks

Chris Flatters
NRAO

November 12, 1993

1 Introduction

NRAO has been asked to write an AIPS task to perform interactive model-fitting for space VLBI data. This program will use the X Window System to display a representation of the model in one window and allow the user to move or stretch components of the model using the mouse while changes to the model visibilities are shown in separate windows. This will require a tighter coupling between AIPS and X11[1] than is supported by the current AIPS system.

Since it is likely that the existence of one X11-based AIPS task will encourage a demand for others, I propose to write an AIPS extension that will make it feasible to write AIPS tasks with X Window System interfaces. Such tasks may be written using the C programming language or C++ and will have access to the AIPS libraries. I have performed a number of experiments to investigate possible approaches to integrating AIPS with X11 and I am confident that this proposal is feasible and can be implemented by a single programmer in a short time. I will describe the proposed extension in section 2 and provide some rationale for it in section 3. I will discuss the amount of manpower required to implement this proposal in section 4.

2 The Proposal

I propose that an enhancement package be distributed separately from AIPS or as an optional part of the AIPS distribution. This package will consist of the following items.

[1] I will indiscriminantly refer to the X Window System by its full name and as X11 throughout this document.

- One or more libraries which provide a ANSI/ISO C interface to a subset of the AIPS library routines.
- One or more libraries providing common utilities for programs based on the OSF/Motif user-interface toolkit.
- A set of imake configuration files which extend the X11 imake system by adding definitions and rules that may be used to build AIPS tasks.
- An imake bootstrap program (analogous to the `xmkmf` command used by the X Window System) that invokes `imake` with the appropriate options for building makefiles for AIPS tasks.
- An installation script that generates the site-dependent configuration files for the imake system using information that is obtained automatically or by quizzing the installer.

The package should be supported on the most common UNIX variants but may not be made available for other operating systems (although the possibility of porting to other operating systems in the future should not be ruled out)..

3 Rationale

In this section I will discuss the package components in more-or-less the order in which they were introduced in Section 2.

3.1 The AIPS/C Interface

The C programming language has the most mature tools for developing X Window System user-interfaces of any language. The only X Consortium library standards are for C libraries (Xlib and Xt) and the most commonly used user-interface toolkits are based on C.

Taken together with the fact that there is an ANSI/ISO standard for the C programming language that ensures code portability between a wide range of compilers, the availability of these tools means that the C programming language is the programming language of choice for writing programs that use the X Window System.

There is no fundamental reason why AIPS tasks should not be written in C (although this is not currently supported) but it is necessary to call the AIPS libraries, which have a FORTRAN 77 interface from C. Not only would it be expensive to duplicate some of the algorithms in the AIPS libraries but it is important to use the AIPS I/O system to access AIPS files so that FORTRAN tasks and C tasks do not clash over access to files. Unfortunately there are two problems with calling FORTRAN libraries from C.

The first problem is that the conventions for calling FORTRAN from C and *vice versa* can vary from compiler to compiler which leads to portability problems. The second is that the conventions for calling FORTRAN from C are rather clumsy. Take the example of writing an AIPS message in C. Under many UNIX systems this would require code that looks like this.

```
const int errMsgLevel = 8;
...
zmsgwr_ ((float *)"An error message", &errMsgLevel);
```

Note that although the message level is not altered by `zmsgwr_` it must be passed by reference to be compatible with FORTRAN argument rules which means that an extra named storage location must be used (`errMsgLevel` in the example above). This is rather unnatural in C and it is easy to make mistakes when doing this. Such mistakes will usually be caught by the compiler but are still frustrating. In addition the introduction of extra variables and constants can make the code cluttered and difficult to read.

A C-language interface library would have the advantage of concentrating all of the C/FORTRAN interface code and its potential portability problems into one place. It would also allow much of the complexity involved in calling FORTRAN from C to be hidden from the programmer.

The C interfaces would also be useable from C++. In the longer term it might be worth redefining the library interfaces in IDL (the CORBA interface definition language) so that they may be used from any language that has an IDL binding.

3.2 The Utility Library

The OSF/Motif user-interface toolkit is now the *de facto* standard user-interface toolkit for the X Window System. It has been submitted for approval as an X/Open standard interface and is the basis for a proposed IEEE standard (the Modular Toolkit Environment or MTE — IEEE draft standard P1295.1). Motif is shipped as a standard operating system component by almost all of the major UNIX vendors[2]. It is, therefore, reasonable to expect that most AIPS/X11 programs will be written using the OSF/Motif toolkit.

Motif applications are expected to behave as specified by the OSF/Motif Style Guide. Unfortunately the OSF/Motif toolkit does not provide all of the facilities required to make application conform to the style guide. The most obvious example is that pressing the `<Help>` key should display context-sensitive help (item 5-13 of the OSF/Motif Level One Certification Checklist) but the OSF/Motif toolkit provides little support for on-line help.

[2]The most prominent exception is Sun, who provide Motif as an optional extra and will integrate it into their standard release in 1994.

Providing a library of missing facilities will not only reduce the amount of programming effort needed to produce a professional-looking application but will encourage uniform behaviour.

Such a library could also hold smaller items that tend to crop up repeatedly in X11/Motif programming. For example many programs need to use `XmNmodifyVerifyCallback` routines to prevent users from typing words into a numeric fields[3].

3.3 The Imake System

The imake system uses the C preprocessor to generate make files using templates for the make rules. It is often used to isolate system and site dependencies into a set of configuration files so that application programmers can supply system-independent imake files rather that require installers to configure complicated make files.

The MIT sample implementation of the X Window System provide an imake configuration that already contains most of the information needed to build AIPS programs that use X11 and most X Window System vendors include this in their distributions. Only a few extra AIPS-specific definitions are needed to compile AIPS tasks and these can be provided by modifying the basic MIT template (`Imake.tmpl`) to include files containing the extra information needed for AIPS programs (this requires adding only a few of code to the template). The `imake` command can then be instructed to use the modified template instead of the original MIT template by setting the appropriate command line flags. A makefile bootstrap program can be used to invoke `imake` with the correct flags rather than expecting a software installer to remember what they are. The bootstrap program would be used in place of the `xmkmf` command used to configure pure X Window System software.

The information that is needed to compile AIPS programs falls into three classes.

1. System-independent information.
2. Information about the operating system and machine being used.
3. Site-specific information

The first two classes of information can be supplied in the standard distribution but the third must be added locally to a site configuration file. The site configuration file can be generated by a script or program that examines the site configuration and asks the person performing the installation about parameters that it can not determine automatically. This will decrease the possibility of errors in the site-specific configuration file.

[3]Motif text entry widgets have the option of calling a programmer-supplied `XmNmodifyVerifyCallback` routine that can check any text that is inserted and veto the change if it finds the new text inappropriate.

The main disadvantage with using the imake system is that some X Window System vendors still do not provide imake or provide broken versions of it. Fortunately the imake system and the X Window System configuration files are available free of charge separately from the MIT sample implementation of X11.

3.4 Portability Restrictions

UNIX systems tend to be fairly uniform in their handling of C/FORTRAN interfaces. Almost all UNIX FORTRAN compilers retain the FORTRAN subroutine calling conventions of the Berkeley portable FORTRAN 77 compiler with a few minor variations (eg. the presence or lack of a trailing underscore added to routine names). This means that the same implementation of the C/AIPS interface library could be used on most UNIX machines. This would make it easy to support most current AIPS installations since UNIX is currently the AIPS platform of choice for most AIPS sites.

The C interface library could be reimplemented for other operating systems (eg. OpenVMS or Windows NT) but there is little incentive to invest manpower in this until there is significant interest in running AIPS on these operating systems. Porting the make procedures to a non-UNIX system would be considerably more work.

For the time being, UNIX-only support seems quite sufficient.

4 Manpower Requirements

The system that I have proposed is actually a repackaging of work that would have to be done to write a single X11-based AIPS task. The advantage of splitting this work out into a separate packae is that it need not be duplicated if further X11-based tasks need to be written. Very little extra manpower is needed to do this if the C/AIPS interface is written incrementally (ie. if it initially contains only those interfaces that are needed by the first X11-based tasks and is extended as the need arises). I would estimate that an initial version of the package would add less than a man-month to the programmer time required to develop the first X11 task with most of the overhead coming from the need to document the support routines and compilation procedures more extensively than would be required if they were part of a single program.